下载APP

06 | 数据同步：主从库如何实现数据一致？

2020-08-17 蒋德钧

Redis核心技术与实战

进入课程 >

讲述：蒋德钧
时长 16:25　大小 15.04M

你好，我是蒋德钧。

前两节课，我们学习了 AOF 和 RDB，如果 Redis 发生了宕机，它们可以分别通过回放日志和重新读入 RDB 文件的方式恢复数据，从而保证尽量少丢失数据，提升可靠性。

不过，即使用了这两种方法，也依然存在服务不可用的问题。比如说，我们在实际使用时只运行了一个 Redis 实例，那么，如果这个实例宕机了，它在恢复期间，是无法服务新来的数据存取请求的。

那我们总说的 Redis 具有高可靠性，又是什么意思呢？其实，这里有两层含义：一是**数据尽量少丢失**，二是**服务尽量少中断**。AOF 和 RDB 保证了前者，而对于后者，Redis 的做

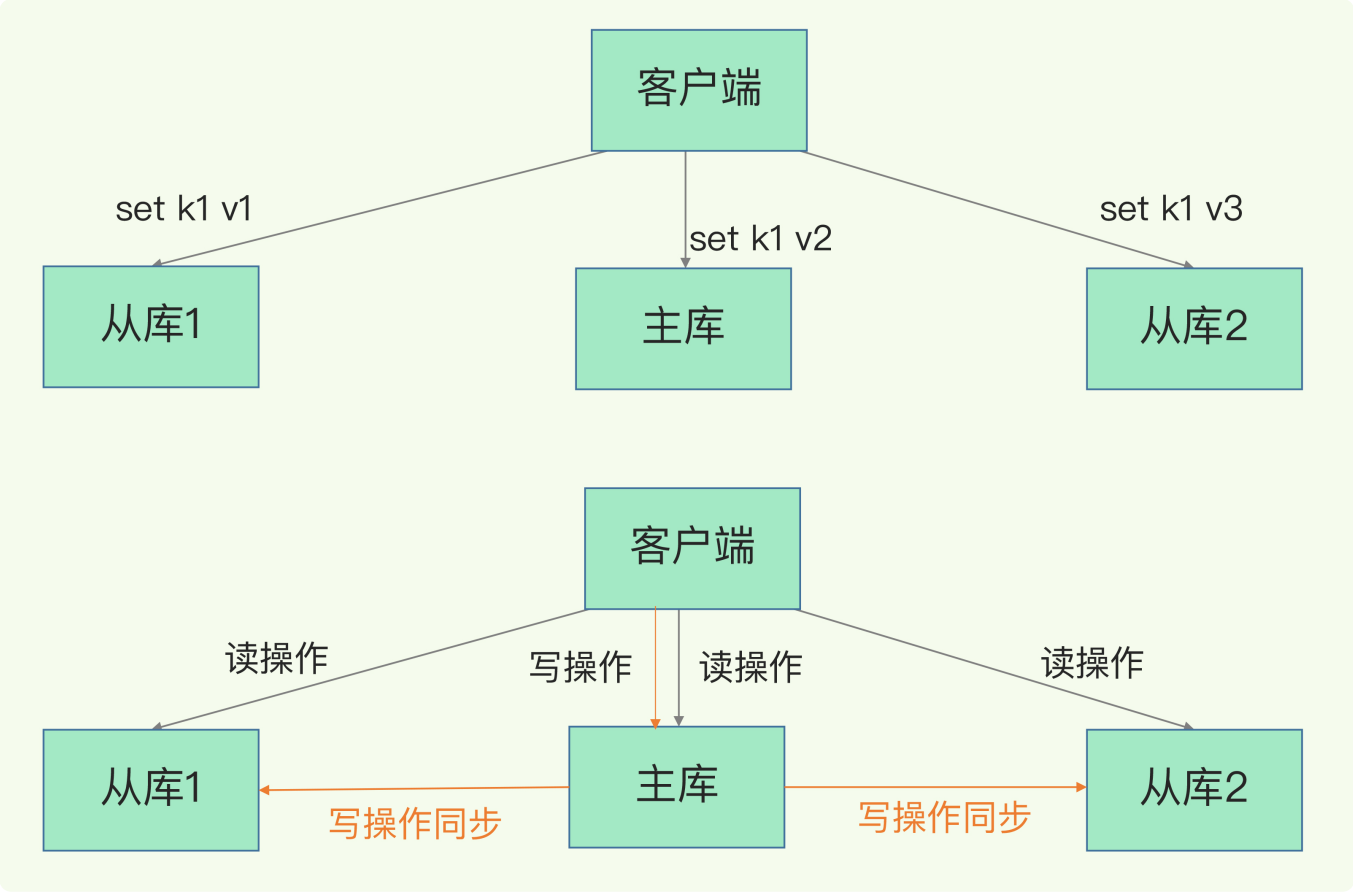
法就是**增加副本冗余量**，将一份数据同时保存在多个实例上。即使有一个实例出现了故障，需要过一段时间才能恢复，其他实例也可以对外提供服务，不会影响业务使用。

多实例保存同一份数据，听起来好像很不错，但是，我们必须要考虑一个问题：这么多副本，它们之间的数据如何保持一致呢？数据读写操作可以发给所有的实例吗？

实际上，Redis 提供了主从库模式，以保证数据副本的一致，主从库之间采用的是读写分离的方式。

读操作：主库、从库都可以接收；

写操作：首先到主库执行，然后，主库将写操作同步给从库。



Redis主从库和读写分离

那么，为什么要采用读写分离的方式呢？

你可以设想一下，如果在上图中，不管是主库还是从库，都能接收客户端的写操作，那么，一个直接的问题就是：如果客户端对同一个数据（例如 k1）前后修改了三次，每一次的修改请求都发送到不同的实例上，在不同的实例上执行，那么，这个数据在这三个实例

上的副本就不一致了（分别是 v1、v2 和 v3）。在读取这个数据的时候，就可能读取到旧的值。

如果我们非要保持这个数据在三个实例上一致，就要涉及到加锁、实例间协商是否完成修改等一系列操作，但这会带来巨额的开销，当然是不太能接受的。

而主从库模式一旦采用了读写分离，所有数据的修改只会在主库上进行，不用协调三个实例。主库有了最新的数据后，会同步给从库，这样，主从库的数据就是一致的。

那么，主从库同步是如何完成的呢？主库数据是一次性传给从库，还是分批同步？要是主从库间的网络断连了，数据还能保持一致吗？这节课，我就和你聊聊主从库同步的原理，以及应对网络断连风险的方案。

好了，我们先来看看主从库间的第一次同步是如何进行的，这也是 Redis 实例建立主从库模式后的规定动作。

主从库间如何进行第一次同步？

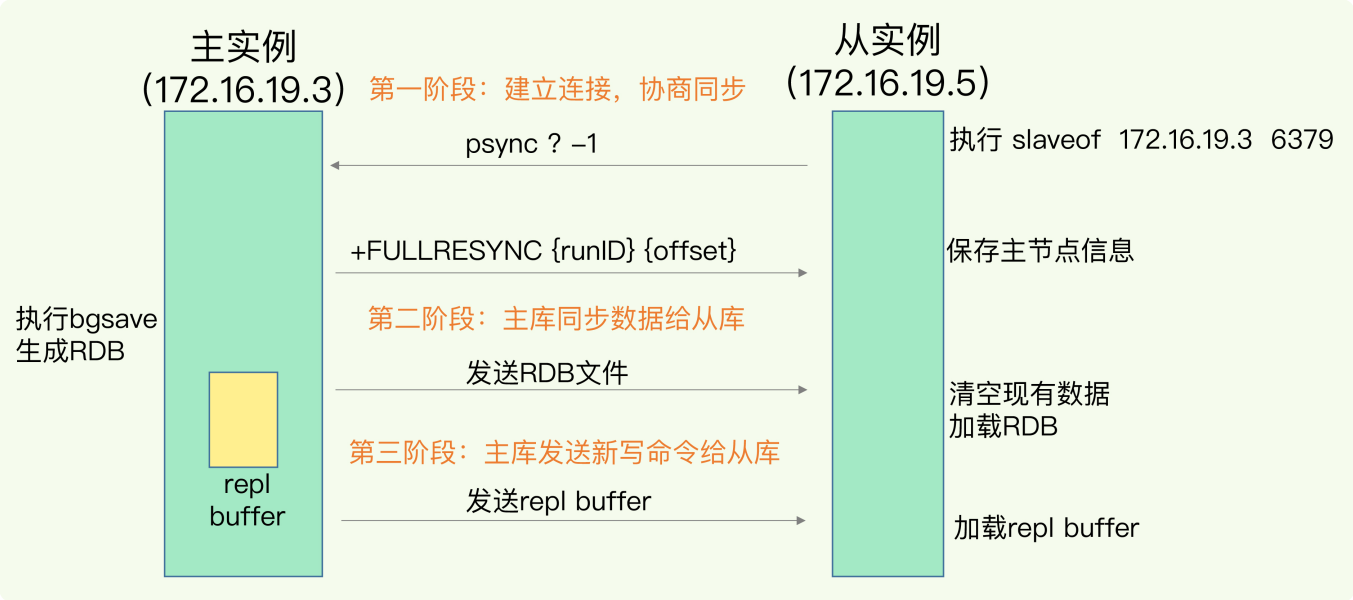
当我们启动多个 Redis 实例的时候，它们相互之间就可以通过 replicaof（Redis 5.0 之前使用 slaveof）命令形成主库和从库的关系，之后会按照三个阶段完成数据的第一次同步。

例如，现在有实例 1（ip：172.16.19.3）和实例 2（ip：172.16.19.5），我们在实例 2 上执行以下这个命令后，实例 2 就变成了实例 1 的从库，并从实例 1 上复制数据：

复制代码

```
1 replicaof  172.16.19.3  6379
```

接下来，我们就要学习主从库间数据第一次同步的三个阶段了。你可以先看一下下面这张图，有个整体感知，接下来我再具体介绍。



主从库第一次同步的流程

第一阶段是主从库间建立连接、协商同步的过程，主要是为全量复制做准备。在这一步，**从库和主库建立起连接，并告诉主库即将进行同步，主库确认回复后，主从库间就可以开始同步了**。

具体来说，从库给主库发送 psync 命令，表示要进行数据同步，主库根据这个命令的参数来启动复制。psync 命令包含了**主库的 runID** 和**复制进度 offset** 两个参数。

> runID，是每个 Redis 实例启动时都会自动生成的一个随机 ID，用来唯一标记这个实例。当从库和主库第一次复制时，因为不知道主库的 runID，所以将 runID 设为“?”。
>
> offset，此时设为 -1，表示第一次复制。

主库收到 psync 命令后，会用 FULLRESYNC 响应命令带上两个参数：主库 runID 和主库目前的复制进度 offset，返回给从库。从库收到响应后，会记录下这两个参数。

这里有个地方需要注意，**FULLRESYNC 响应表示第一次复制采用的全量复制，也就是说，主库会把当前所有的数据都复制给从库**。

在第二阶段，**主库将所有数据同步给从库。从库收到数据后，在本地完成数据加载**。这个过程依赖于内存快照生成的 RDB 文件。

具体来说，主库执行 bgsave 命令，生成 RDB 文件，接着将文件发给从库。从库接收到 RDB 文件后，会先清空当前数据库，然后加载 RDB 文件。这是因为从库在通过 replicaof 命令开始和主库同步前，可能保存了其他数据。为了避免之前数据的影响，从库需要先把当前数据库清空。

在主库将数据同步给从库的过程中，主库不会被阻塞，仍然可以正常接收请求。否则，Redis 的服务就被中断了。但是，这些请求中的写操作并没有记录到刚刚生成的 RDB 文件中。为了保证主从库的数据一致性，主库会在内存中用专门的 replication buffer，记录 RDB 文件生成后收到的所有写操作。

最后，也就是第三个阶段，主库会把第二阶段执行过程中新收到的写命令，再发送给从库。具体的操作是，当主库完成 RDB 文件发送后，就会把此时 replication buffer 中的修改操作发给从库，从库再重新执行这些操作。这样一来，主从库就实现同步了。

主从级联模式分担全量复制时的主库压力

通过分析主从库间第一次数据同步的过程，你可以看到，一次全量复制中，对于主库来说，需要完成两个耗时的操作：生成 RDB 文件和传输 RDB 文件。

如果从库数量很多，而且都要和主库进行全量复制的话，就会导致主库忙于 fork 子进程生成 RDB 文件，进行数据全量同步。fork 这个操作会阻塞主线程处理正常请求，从而导致主库响应应用程序的请求速度变慢。此外，传输 RDB 文件也会占用主库的网络带宽，同样会给主库的资源使用带来压力。那么，有没有好的解决方法可以分担主库压力呢？

其实是有的，这就是“主 - 从 - 从”模式。

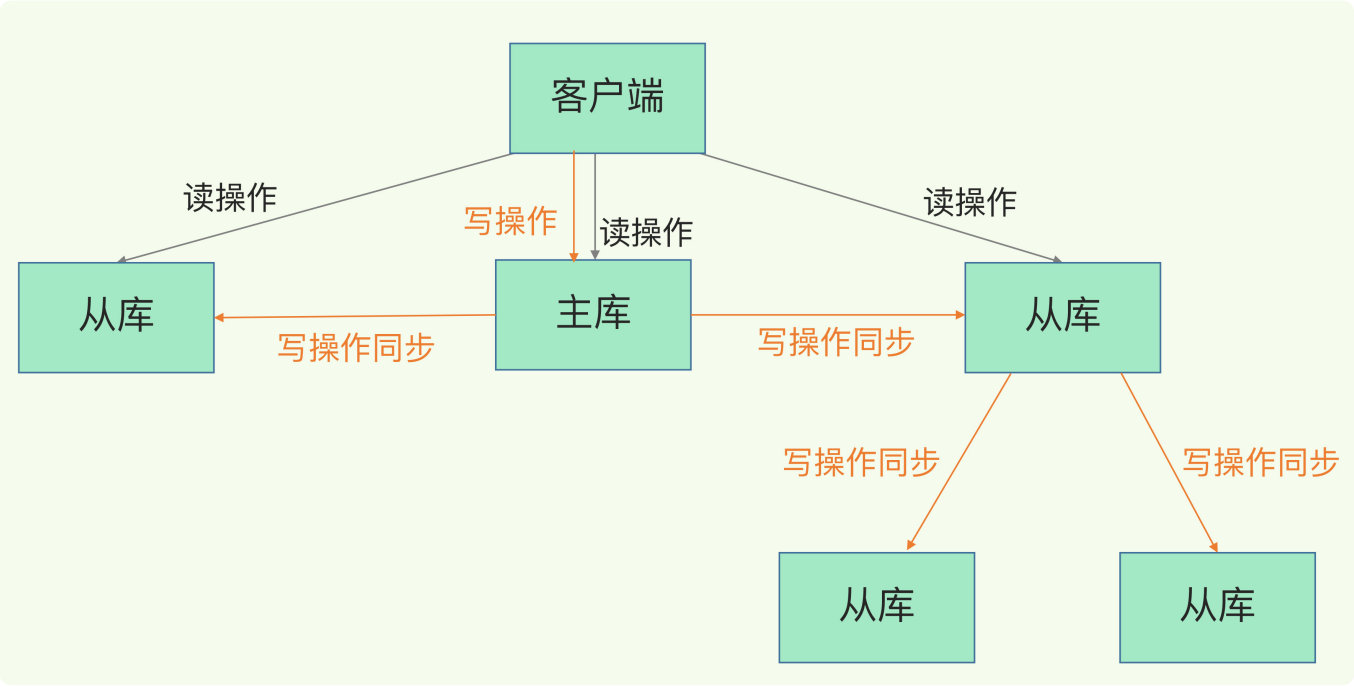
在刚才介绍的主从库模式中，所有的从库都是和主库连接，所有的全量复制也都是和主库进行的。现在，我们可以**通过“主 - 从 - 从”模式将主库生成 RDB 和传输 RDB 的压力，以级联的方式分散到从库上**。

简单来说，我们在部署主从集群的时候，可以手动选择一个从库（比如选择内存资源配置较高的从库），用于级联其他的从库。然后，我们可以再选择一些从库（例如三分之一的从库），在这些从库上执行如下命令，让它们和刚才所选的从库，建立起主从关系。

复制代码

```
1 replicaof 所选从库的IP 6379
```

这样一来，这些从库就会知道，在进行同步时，不用再和主库进行交互了，只要和级联的从库进行写操作同步就行了，这就可以减轻主库上的压力，如下图所示：



级联的“主-从-从”模式

好了，到这里，我们了解了主从库间通过全量复制实现数据同步的过程，以及通过“主 - 从 - 从”模式分担主库压力的方式。那么，一旦主从库完成了全量复制，它们之间就会一直维护一个网络连接，主库会通过这个连接将后续陆续收到的命令操作再同步给从库，这个过程也称为**基于长连接的命令传播**，可以避免频繁建立连接的开销。

听上去好像很简单，但不可忽视的是，这个过程中存在着风险点，最常见的就是**网络断连或阻塞**。如果网络断连，主从库之间就无法进行命令传播了，从库的数据自然也就没办法和主库保持一致了，客户端就可能从从库读到旧数据。

接下来，我们就来聊聊网络断连后的解决办法。

主从库间网络断了怎么办？

在 Redis 2.8 之前，如果主从库在命令传播时出现了网络闪断，那么，从库就会和主库重新进行一次全量复制，开销非常大。

从 Redis 2.8 开始，网络断了之后，主从库会采用增量复制的方式继续同步。听名字大概就可以猜到它和全量复制的不同：全量复制是同步所有数据，而增量复制只会把主从库网络断连期间主库收到的命令，同步给从库。

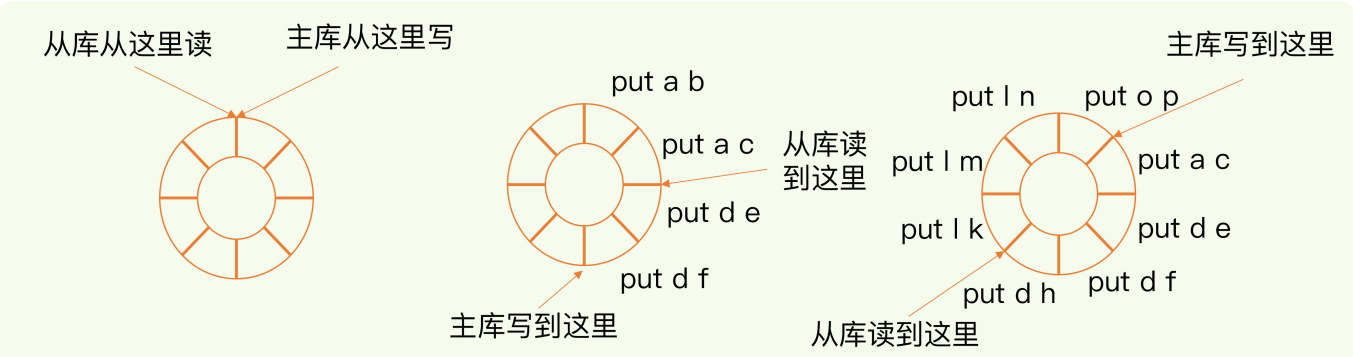
那么，增量复制时，主从库之间具体是怎么保持同步的呢？这里的奥妙就在于 repl_backlog_buffer 这个缓冲区。我们先来看下它是如何用于增量命令的同步的。

当主从库断连后，主库会把断连期间收到的写操作命令，写入 replication buffer，同时也会把这些操作命令也写入 repl_backlog_buffer 这个缓冲区。

repl_backlog_buffer 是一个环形缓冲区，**主库会记录自己写到的位置，从库则会记录自己已经读到的位置。**

刚开始的时候，主库和从库的写读位置在一起，这算是它们的起始位置。随着主库不断接收新的写操作，它在缓冲区中的写位置会逐步偏离起始位置，我们通常用偏移量来衡量这个偏移距离的大小，对主库来说，对应的偏移量就是 master_repl_offset。主库接收的新写操作越多，这个值就会越大。

同样，从库在复制完写操作命令后，它在缓冲区中的读位置也开始逐步偏移刚才的起始位置，此时，从库已复制的偏移量 slave_repl_offset 也在不断增加。正常情况下，这两个偏移量基本相等。



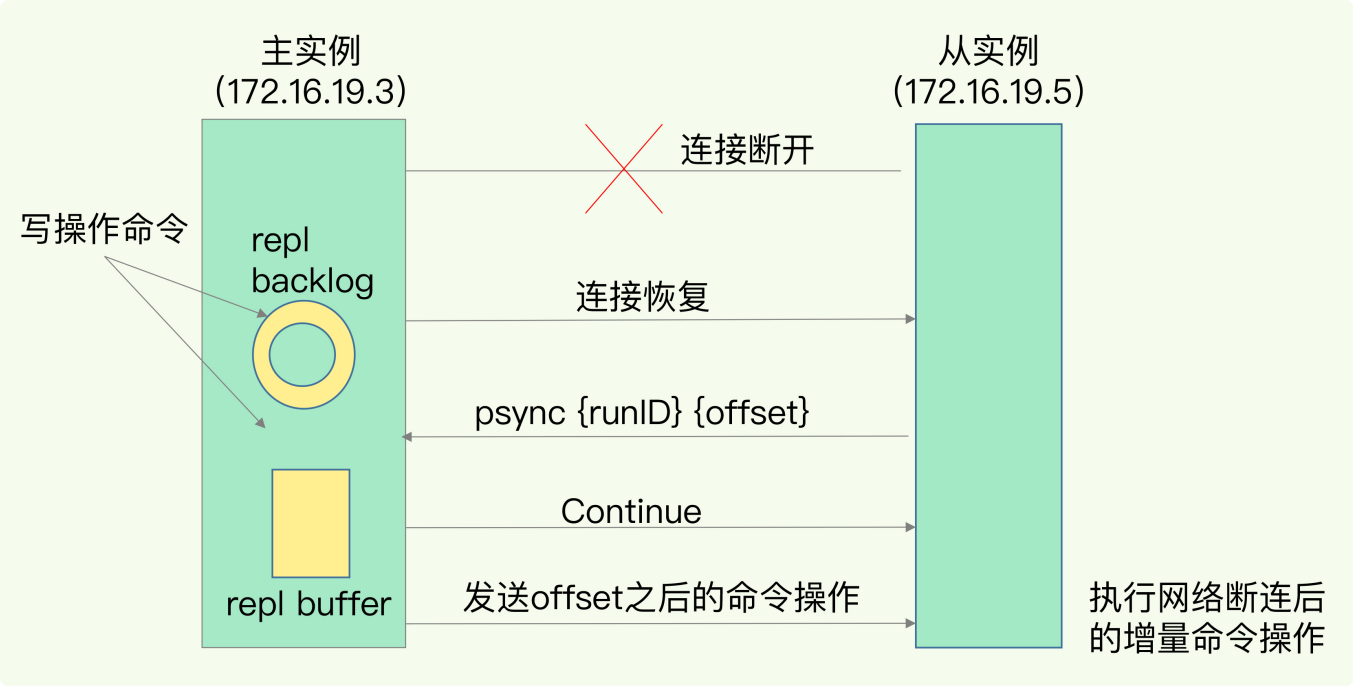
Redis repl_backlog_buffer的使用

主从库的连接恢复之后，从库首先会给主库发送 psync 命令，并把自己当前的 slave_repl_offset 发给主库，主库会判断自己的 master_repl_offset 和 slave_repl_offset 之间的差距。

在网络断连阶段，主库可能会收到新的写操作命令，所以，一般来说，master_repl_offset 会大于 slave_repl_offset。此时，主库只用把 master_repl_offset 和 slave_repl_offset 之间的命令操作同步给从库就行。

就像刚刚示意图的中间部分，主库和从库之间相差了 put d e 和 put d f 两个操作，在增量复制时，主库只需要把它们同步给从库，就行了。

说到这里，我们再借助一张图，回顾下增量复制的流程。



Redis增量复制流程

不过，有一个地方我要强调一下，因为 repl_backlog_buffer 是一个环形缓冲区，所以在缓冲区写满后，主库会继续写入，此时，就会覆盖掉之前写入的操作。**如果从库的读取速度比较慢，就有可能导致从库还未读取的操作被主库新写的操作覆盖了，这会导致主从库间的数据不一致。**

因此，我们要想办法避免这一情况，一般而言，我们可以调整 **repl_backlog_size** 这个参数。这个参数和所需的缓冲空间大小有关。缓冲空间的计算公式是：缓冲空间大小 = 主库写入命令速度 * 操作大小 - 主从库间网络传输命令速度 * 操作大小。在实际应用中，考虑到可能存在一些突发的请求压力，我们通常需要把这个缓冲空间扩大一倍，即 repl_backlog_size = 缓冲空间大小 * 2，这也就是 repl_backlog_size 的最终值。

举个例子，如果主库每秒写入 2000 个操作，每个操作的大小为 2KB，网络每秒能传输 1000 个操作，那么，有 1000 个操作需要缓冲起来，这就至少需要 2MB 的缓冲空间。否则，新写的命令就会覆盖掉旧操作了。为了应对可能的突发压力，我们最终把 repl_backlog_size 设为 4MB。

这样一来，增量复制时主从库的数据不一致风险就降低了。不过，如果并发请求量非常大，连两倍的缓冲空间都存不下新操作请求的话，此时，主从库数据仍然可能不一致。

针对这种情况，一方面，你可以根据 Redis 所在服务器的内存资源再适当增加 repl_backlog_size 值，比如说设置成缓冲空间大小的 4 倍，另一方面，你可以考虑使用切片集群来分担单个主库的请求压力。关于切片集群，我会在第 9 讲具体介绍。

小结

这节课，我们一起学习了 Redis 的主从库同步的基本原理，总结来说，有三种模式：全量复制、基于长连接的命令传播，以及增量复制。

全量复制虽然耗时，但是对于从库来说，如果是第一次同步，全量复制是无法避免的，所以，我给你一个小建议：**一个 Redis 实例的数据库不要太大**，一个实例大小在几 GB 级别比较合适，这样可以减少 RDB 文件生成、传输和重新加载的开销。另外，为了避免多个从库同时和主库进行全量复制，给主库过大的同步压力，我们也可以采用“主 - 从 - 从”这一级联模式，来缓解主库的压力。

长连接复制是主从库正常运行后的常规同步阶段。在这个阶段中，主从库之间通过命令传播实现同步。不过，这期间如果遇到了网络断连，增量复制就派上用场了。我特别建议你留意一下 repl_backlog_size 这个配置参数。如果它配置得过小，在增量复制阶段，可能会导致从库的复制进度赶不上主库，进而导致从库重新进行全量复制。所以，通过调大这个参数，可以减少从库在网络断连时全量复制的风险。

不过，主从库模式使用读写分离虽然避免了同时写多个实例带来的数据不一致问题，但是还面临主库故障的潜在风险。主库故障了从库该怎么办，数据还能保持一致吗，Redis 还能正常提供服务吗？在接下来的两节课里，我会和你具体聊聊主库故障后，保证服务可靠性的解决方案。

每课一问

按照惯例，我给你提一个小问题。这节课，我提到，主从库间的数据复制同步使用的是 RDB 文件，前面我们学习过，AOF 记录的操作命令更全，相比于 RDB 丢失的数据更少。那么，为什么主从库间的复制不使用 AOF 呢？

好了，这节课就到这里，如果你觉得有收获，欢迎你帮我把今天的内容分享给你的朋友。

提建议

© 版权归极客邦科技所有，未经许可不得传播售卖。 页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 05 | 内存快照：宕机后，Redis如何实现快速恢复？

精选留言 (24)

写留言

Kaito

2020-08-17

主从全量同步使用RDB而不使用AOF的原因：

1、RDB文件内容是经过压缩的二进制数据（不同数据类型数据做了针对性优化），文件很小。而AOF文件记录的是每一次写操作的命令，写操作越多文件会变得很大，其中还包括很多对同一个key的多次冗余操作。在主从全量数据同步时，传输RDB文件可以尽量降...

展开

16 25

Geek_121747
2020-08-17

每课一问：1.因为AOF文件比RDB文件大，网络传输比较耗时，
2.从库在初始化数据时，RDB文件比AOF文件执行更快

7

Darren
2020-08-17

上一条评论有误，已经删除，请通过本条评论

repl_backlog_size这个参数很重要，因为如果满了，就需要重新全量复制，默认是1M，所以之前网上就流传1个段子，如果一个公司说自己体量如何大，技术多么牛，要是repl_backlog_size参数是默认值，基本可以认为要不业务体量吹牛逼了，要不就没有真正的技...

展开

4

test
2020-08-17

课后题：aof比rdb大，rdb加载起来比aof快。

展开

2

Monday
2020-08-17

repl_backlog_buffer是以从库为粒度保存的吧，也就说在一主多从的情况下，主会分别保存多每个从库的处理位置吧。
思考题：
1、rdb比aof通常情况下会大些，
2、传播起来更耗带宽等资源...

展开

1

Jackey
2020-08-17

印象中Redis在repl_backlog_buffer写满了之后会触发一次全量复制，以此来保证被覆盖的数据不丢失。不知道对不对

展开

1

tt
2020-08-17

我觉得之所以主从库间的数据复制同步使用的是 RDB 文件，是因为主从复制要解决的是不同节点之间状态的一致性问题，RDB文件的内容正好是某一个时间点主库的状态

AOF文件则不同，其内容本质上是操作而不是状态，内容存在大量冗余。主从复制本质上是一个最终一致性的过程，为了减少不一致的时间窗口的大小，采用RDB文件是最合适...

展开

1

王聪
2020-08-17

看了课程内容,不看评论绝对是一种损失

展开

Kyushu
2020-08-17

除了从库命令读取比较慢以外，如果网络断连再恢复造成了repl_backlog_buffer的覆盖也就永远的不一致了吧，这样也只能通过调大他的值决定吗

展开

Frank
2020-08-17

1. 相同数据下，AOF 文件比 RDB 更大，因此需要的网络带宽更多；
2. 在恢复数据时，使用RDB更快。
3. 如果使用AOF文件来同步相对来说丢的数据更少，但是不表示不丢数据。即也需要第三个阶段来保证数据的一致性。因此相对来说使用RDB开销更小些。

展开

yyl

2020-08-17

解答：主从同步采用RDB的原因是由于采用RDB，从节点恢复效率更好，能够更快地对外提供服务，分担主节点的读压力。
如果采用AOF日志，从节点需要逐条执行命令，恢复效率低

展开

啦啦啦
2020-08-17

从库在执行 replication buffer里面的命令的时候，这时候主库不又会产生新的数据了吗？这样下去不就没完没了了吗

展开

醉、随风😳 😳
2020-08-17

我有个问题，为什么一定要同时存在repl_backlog_buffer和replication buffer两个缓冲区，而且repl_backlog_buffer确要定义成环形的。我觉得是否可以定义成一个链表，然后主库的offset与从库的offset都指向链表中某个元素，redis定时清理从库offset之前的数据，这样不是做到了既能增量同步，也不用担心因为并发过大造成的同步数据丢失。

展开

猿人谷
2020-08-17

补充一个知识点：主从模式下，Redis能否读到过期键？
当 Redis 运行在主从模式下时，从库不会进行过期扫描，从库对过期键的处理是被动的。也就是即使从库中的 key 过期了，如果有客户端访问从库时，依然可以得到 key 对应的值，像未过期的键值对一样返回。

展开

那时刻
2020-08-17

请问老师，FULLRESYNC命令里的复制进度offset指的是repl_backlog_buffer里master_repl_offset吗？

yayiyaya
2020-08-17

使用 主 -从 -从 模式， 客户端的请示是怎么到达第二个从库的？ 不太理解

阿卡牛
2020-08-17

主库和从库可以布署在不同的地理位置吗

展开

1

QFY
2020-08-17

课后题：
aof记录的是命令，如果是首次主从全量复制，而且如果aof文件还没被重写会存在对一个key的反复操作，那么效率会比较低

疑问：...

展开

tt
2020-08-17

老师，还有一个问题，主从同步时，生成RDB快照是由bgsave进程完成的，那主库发送RDB文件内容和从库接收数据各是有哪些线程完成的呢？也是主线程么？

展开

Mr.蜜
2020-08-17

我有一个疑问，环形缓冲期再大，也会出问题，那么如果遇到这类问题，导致数据不同步怎么处理？比方说，一个从库长断网以后，长时间没有联网处理。

展开

作者回复: 没错，环形缓冲区再大，在某些时候，就如你所说的从库长期断网时，也会出问题。

其实从库正常情况下会每秒给主库发送一个replconf ack命令，主库会根据这个命令的达到时间判断和从库的连网情况。如果距离最后一次ack命令收到的时间已经超过了repl_timeout时间，就会和从库断开连接了。

从库再和主库连接时，会发送自己的复制进度，如果要复制内容在缓冲区中已经被覆盖了，那么就不再做增量复制了，而是进行全量复制。